Serna, M. E. (2009). La ingeniería de sistemas y su evolución hacia la arquitectura de sistemas. Revista Digital Lámpsakos, No. 2, pp. 96-105.

LA INGENIERÍA DE SISTEMAS Y SU EVOLUCIÓN HACIA LA ARQUITECTURA DE SISTEMAS

Edgar Serna Montoya
Grupo de investigación SISCO. Funlam, Colombia.
gruposisco@gmail.com

(Artículo de TRADUCCIÓN) (Recibido el 3 de agosto. Aprobado el 10 de octubre de 2009)

RESUMEN
Muchas de las empresas modernas entendieron que sus antiguas unidades de sistemas ya no son funcionales, y comienzan a subdividirlas en dos grupos de trabajo diferenciadores: el encargado de la infraestructura y el de los denominados "arquitectos de sistemas". Esta decisión lógica la inspira la actual evolución de la Ingeniería de Sistemas que, como área de conocimiento, genera los mismos subgrupos como agentes para formación. Además, la evolución y complejidad de los sistemas de información en medio de la sociedad del conocimiento, con exigencias y expectativas muy complejas, también determinan la necesidad de esta especialización. En este documento, una traducción casi literal de un *white paper* que publicó la empresa Quidnunc -www.quidnunc.com consultado en abril del año 2000- especializada en gestión de configuración, se detalla la importancia de esta división y las pautas a seguir a la hora de diseñar la arquitectura de sistemas de una empresa.

Palabras clave: Arquitectura de sistemas, Ingeniería de sistemas, Arquitecto de sistemas, infraestructura.

INTRODUCCIÓN

Entendemos por arquitectura en un proyecto informático a la disposición conjunta y ordenada de elementos del software y del hardware con el objetivo de cumplir una determinada función. No es difícil comprender que al mezclar arquitecturas distintas e inconsistentes sin ningún tipo de orden o planificación, el proyecto se puede convertir fácilmente en inmanejable, tanto o más cuanto mayor sea el tamaño del mismo.

La mayoría de las organizaciones tradicionalmente favorecen -de forma planificada o no- unas configuraciones concretas. La arquitectura de cada empresa debería describir estas configuraciones, así como el entorno que facilite la creación de nuevas funcionalidades que encajen en ella: directivas, componentes de software reutilizables, herramientas, entre otras. Para facilitar que las nuevas funcionalidades implementadas en la nueva arquitectura sean consistentes con el sistema actual y sus posibles modificaciones futuras, es necesario conocer dicha arquitectura, pero es mucho más importante conocer la arquitectura operativa y organizacional de la empresa.

Una distinción importante es la que existe entre la arquitectura de una aplicación sencilla -micro arquitectura- y la que existe entre y a través de las distintas aplicaciones -macro arquitectura-, por mucho, más compleja e importante.

DIVIDE Y VENCERÁS

¿Cuál es el papel de la arquitectura en una organización? Imaginemos una organización que consta de cuatro capas: la capa superior, formada por las actividades propias de la organización; debajo de éstas, las aplicaciones informáticas que las soportan y facilitan; más abajo de las aplicaciones, la arquitectura que facilita que se desarrollen y ejecuten; y en último lugar, la infraestructura del hardware o las redes físicas. Esta subdivisión facilita determinar el papel que desempeña la arquitectura al interior de una organización: cada capa actúa como cliente de la capa inferior a ella y como servidor de la capa superior. Los arquitectos no deben malgastar su tiempo en temas relacionados con la infraestructura, como el sistema operativo; la mejor forma de separar la arquitectura de la infraestructura es pensar en el esquema de cuatro capas mencionado: la infraestructura debe de dar soporte a la arquitectura, ya que mezclar conceptos de una y otra capa es un error muy común en las organizaciones.

Un error en el que no debe de caer un arquitecto de sistemas es en ser demasiado preceptivo: introducir demasiadas normas que creen en excesiva rigidez, generará problemas en el desarrollo de aplicaciones. Un buen arquitecto de sistemas debe tener siempre en mente que su finalidad principal es permitir la creación de aplicaciones, a la vez que facilitar la creatividad y la innovación de los creadores de las mismas.

La arquitectura de sistemas en los tiempos en los que sólo existían los grandes computadores era muy sencilla: existía un lugar para cada cosa y cada cosa tenía su lugar adecuado. Con el paso de los años, y siempre en busca de una mayor flexibilidad, se introdujeron estructuras cada vez más complejas: arquitectura cliente/servidor, arquitectura a tres capas, *message brokers*, *data warehouses*, objetos distribuidos, arquitectura *web*... Un buen arquitecto debe empezar recordando que su trabajo es hacer la vida más fácil a los desarrolladores, y no al revés.

Existe otra idea que subyace tras todos los enfoques: especialización. Dividir los problemas en sus partes constituyentes y resolverlas separadamente con equipos de especialistas centrados en una sola área. La especialización deja dos interrogantes sin respuesta: cómo dividir los sistemas para que puedan ser definidos separadamente y cómo unirlos posteriormente para formar un todo homogéneo. Estos son los principales retos de la moderna arquitectura de sistemas.

ARQUITECTURA EVOLUTIVA

Y, desde el punto de vista de la empresa, ¿qué características debería reunir la arquitectura de sistemas? Si mejorar los procesos y las aplicaciones genera un mejor rendimiento de la empresa, y si para mejorar las aplicaciones necesitamos mejorar los sistemas, entonces la arquitectura de sistemas debe ser el vehículo de desarrollo para ambos. En la práctica, la arquitectura de los sistemas actuales constituye, en muchos casos, grandes obstáculos para los dos.

A principios de los 90 la arquitectura de sistemas no iba más allá de simple planificación: se define una arquitectura objetivo y se idea una estrategia y una planificación para completarla dentro de unos plazos determinados. La principal ventaja de este enfoque es que la hace comprensible a los ejecutivos, pues es similar a la forma en que tienen que dirigir sus negocios. El principal inconveniente es que no funciona, ya que comienza por la definición de una arquitectura objetivo y esto es un error. El único objetivo que debe tener en mente el arquitecto de sistemas es el de la organización para la que trabaja, si no tarde o temprano entrará en conflicto con él. La arquitectura del sistema debe de ser lo suficientemente flexible como para adaptarse a los cambios de objetivos organizacionales, esta es la clave principal para asegurar su longevidad.

La segunda clave a tener en cuenta es la que proporciona la mejor forma de medir la bondad de una arquitectura: la forma en que sustenta las aplicaciones que a la vez sustentan la organización. La mejor forma de verlo es estudiar, dada una nueva funcionalidad necesaria para la empresa, cómo la arquitectura del sistema facilita su desarrollo e integración con el resto de las aplicaciones.

Los elementos claves que debe cumplir la arquitectura, para facilitar el desarrollo de nuevas aplicaciones, son: tener directivas claramente definidas, no rígidas ni dictatoriales en cuanto al uso de determinadas tecnologías o fabricantes; favorecer el uso de aplicaciones que posean una funcionalidad base y sean personalizables por el usuario; y facilitar el uso y desarrollo de componentes y *plug-ins* y aplicaciones que los admitan. Este enfoque permite, en la mayoría de los casos, encontrar la forma más rápida y sencilla de desarrollar una nueva funcionalidad para el sistema, en los casos en los que lo más importante es tener una aplicación que haga lo que queremos y no tener la mejor aplicación que haga lo que no queremos. Actualmente, en la práctica, esta es la solución que se necesita en la mayoría de los casos, y son los principios del enfoque conocido como evolutivo.

El modelo evolutivo al que la arquitectura del sistema se adapta paso a paso surge del concepto de dependencia, en el que cada uno de ellos se basa en los anteriores para

perfeccionarse y evolucionar en cada momento, sin seguir un plan maestro pero de acuerdo con una "evolución natural". Esta evolución posee dos elementos cruciales: un método para producir variantes -la reproducción- y un método para elegir la mejor entre ellas -la supervivencia de los más fuertes.

Los métodos evolutivos también son populares en el desarrollo de software a medida: las aplicaciones suelen construirse mediante una serie de pasos consecutivos y no de una sola vez, lo que reduce considerablemente el riesgo de fallos y el costo de desarrollo. En este caso y dado que no existe variación ni selección, el término "evolución" no se aplica adecuadamente, y por el contrario el término más adecuado sería "desarrollo incremental".

Igual que en la evolución natural, las variantes en el mundo de la informática son abundantes y sólo los sistemas más abiertos sobreviven. Es fundamental crear un entorno que propicie la tecno-diversidad en la arquitectura del sistema, y una excepción a esta teoría la constituyen, en sí mismos, los *mainframes*, que han resistido más de lo que se podía esperar, incluso luego de la epidemia del año 2000. El problema es cómo crear un entorno que facilite la tecno-diversidad, en el que las arquitecturas preestablecidas sobrevivan donde sea necesario, pero sin bloquear el paso a los nuevos esquemas.

Las organizaciones con arquitecturas evolutivas poseen ciertos rasgos en común:

- Prefieren las directivas a los estándares. Los estándares reales son minimistas y usualmente "de hecho" como Windows; las directivas pueden obviarse si existe una razón lo suficientemente buena. La mayoría de las organizaciones mantienen demasiados de ellos, motivados por la reducción de costos -por ejemplo, mantienen UNIX en el *back-end* para minimizar el costo de reeducación-, pero fallan al ignorar el costo que supone forzar a determinadas aplicaciones que requieren en realidad tomar una línea diferente. La flexibilidad es una necesidad fundamental en la arquitectura de los sistemas modernos.
- Usan tecnología orientada a componentes. La historia de la ingeniería de sistemas describe un viaje inexorable hacia la especialización. Los componentes de hoy son más flexibles y es posible ya reutilizar el software que prometiera desde hace tiempo la programación orientado por objetos.
- Juzgan la arquitectura del sistema desde el punto de vista del usuario.
- Invierten en infraestructura. Ahorrar gastos en hardware o comunicaciones es a menudo un falso ahorro: un efecto de los días en que estas cosas eran caras. Ahorrar dinero ahora traerá otros gastos posteriores.
- Reflejan la arquitectura de su organización en la arquitectura del sistema. Por ejemplo, organizaciones centralizadas necesitan un sistema centralizado, mientras que organizaciones descentralizadas se adaptan mejor a sistemas distribuidos. Esto es una directiva más que una regla.
- A la hora de elegir entre distintas aplicaciones toman como principales criterios la facilidad de uso y el impacto en el negocio.
- Eliminan los proyectos fallidos o débiles rápidamente. Cada dólar invertido en un proyecto débil o fallido es un dinero malgastado dos veces: aprende la lección, evita recriminaciones y corrige el problema.
- Valoran el capital intelectual. El principal activo de un departamento de arquitectura de sistemas son las personas y los procedimientos que conocen o desarrollan. Es preciso cuidar apropiadamente esos aportes.
- Evitan innovaciones.

Estos puntos no pretenden ser una línea de actuación para beneficiar una arquitectura evolutiva, son una lista de observaciones procedente de organizaciones que manejan arquitecturas con estas características.

LA REVOLUCIÓN DE LOS COMPONENTES

Dos décadas después de comenzar a recorrer el camino, por fin se tiene la tecnología suficiente para crear y ensamblar componentes que otras ramas de la ingeniería disfrutan desde hace mucho tiempo. Hasta hace poco la industria del software atravesaba una grave crisis: las

empresas demandaban desarrollos cada vez más y más complejos, en plazos de tiempo cada vez más ajustados, y a precios más competitivos. La actividad de desarrollar software es ya de por si lo suficientemente lenta y frustrante como para tener que aguantar esto. En la historia se encuentran otros sectores de la industria que atravesaron por estos problemas: el ejemplo más claro lo representa la industria automovilística. Actualmente el trabajo en las plantas donde se construyen los autos se limita al ensamblar componentes; estos componentes se suministran listos para que los proveedores, especializados en la fabricación de los mismos, realicen un montaje para el que, posiblemente, no tengan idea de los conocimientos necesarios para hacer componentes distintos. La tecnología de componentes va de la mano de la especialización.

Pero la especialización no fue inventada por la industria del automóvil, la naturaleza la usa desde hace millones de años. Los seres vivos se constituyen de órganos, y cada uno es un conjunto de células altamente especializadas. La ventaja de este esquema es cada uno de ellos puede desenvolverse sin interferencia de los otros. Volviendo al ejemplo de la planta de ensamble de automóviles, las interfaces y especificaciones de todos sus componentes se acuerdan de antemano para que no haya sorpresas durante el proceso. Estas interfaces entre componentes se definen de la forma más simple posible para acelerar el tiempo de montaje. En cualquier caso, en la industria del software se tienen otras peculiaridades: la replicación masiva nunca ha sido problema, el principal objetivo es la personalización masiva: disponer de una amplia variedad de productos basados en un esqueleto común.

LA LEY DE MOORE EN EL SOFTWARE

En esencia, un componente de software es una pieza que realiza funciones bien definidas y posee una interfaz bien definida. Claros ejemplos de los primeros componentes son por ejemplo los VBX, introducidos por Visual Basic, o los *plug-ins,* de Photoshop. Fueron pasos importantes, pero aún tenían dos defectos importantes por superar: sólo servían para un producto específico y, por tanto, de valor limitado, y se concibieron como ampliaciones de la aplicación original, mientras que en las verdaderas aplicaciones basadas en componentes, éstos constituyen casi la totalidad de la aplicación.

Las tecnologías recientes van más allá: *Actives*, *Java Beans*, componentes desarrollados con herramientas que cumplen las especificaciones CORBA, SAP, etc. Estas tecnologías potencian los dos principales beneficios de la tecnología de componentes:

- La división en componentes reduce la complejidad, permite la reutilización y acelera el proceso de ensamblaje de software.
- Los creadores de componentes pueden especializarse creando objetos cada vez más complejos y de mayor calidad.
- La interoperabilidad entre componentes de distintos fabricantes aumenta la competencia, reduce los costos y facilita la construcción de estándares. El software se hace cada vez más rápido, de mejor calidad y a menor costo.
- La principal implicación para la industria del software es que se dividirá en dos: fabricantes y ensambladores de componentes.

EL SECRETO ES... EL EMPAQUETADO

Si los componentes existen desde principios de los años 90 ¿por qué ahora suponen una revolución? Porque los beneficios son alcanzables sólo ahora, cuando la tecnologías para empaquetarlos alcanza la suficiente madurez. *Actives* y *Java Beans* son buenos ejemplos: ambos proporcionan los contenedores donde depositar el código, de forma que pueden ser manejados sin importar el lenguaje en el que están escritos.

El empaquetado del código tardó tanto en desarrollarse debido a que va en contra de la norma de la mayoría de los programadores quienes persiguen la eficiencia del código por encima de la eficiencia en el desarrollo. En los principios de la informática las máquinas eran caras y los programadores baratos, de forma que eran "programados" para conseguir los mejores rendimientos con sus desarrollos, y la idea de colocar capas de código innecesario, con el único propósito de facilitar el desarrollo de aplicaciones, parecía impensable.

Hoy, por el contrario, las máquinas son baratas y la gente que sabe trabajar con ellas muy cara. Entonces aparecieron las técnicas orientadas por objetos: colocar datos y funciones juntas para formar objetos, fue un gran paso sin el que ninguna de las tecnologías de empaquetado de componentes actuales hubiera prosperado. La orientación por objetos es el mayor paradigma que jamás existió en el mundo de la informática. No obstante, los primeros intentos de empaquetado de código fueron un fracaso: los desarrolladores no podían creer que el envoltorio fuese más complejo que el código que contenía. Actualmente, el empaquetado de código sigue siendo complejo y arrastra considerables problemas: aumenta considerablemente el tamaño de los programas, empeora el rendimiento de éstos y hace consumir más recursos en las máquinas donde se ejecutan. Exactamente los mismos problemas que se les atribuyen a las GUI y a pesar de ello prácticamente todo el mundo usa alguna.

A pesar de estos problemas las ventajas superaron a los inconvenientes y, aunque las tecnologías de empaquetado de código siguen siendo muy complejas de construir, alcanzaron un alto grado de facilidad de uso, de forma que los desarrolladores dedicados a la fabricación de componentes sólo tienen que ocuparse de desarrollar cada vez mejores componentes, sin preocuparse de nada más.

A pesar de que hoy existen diversas técnicas de empaquetado de componentes, con grandes diferencias según de donde procedan, también tienen importantes similitudes:

- Transparencia en cuanto a la localización, lo que permite usar un componente sin la preocupación de dónde se encuentra físicamente, incluso si éste se cambia de sitio.
- Definición de la interfaz. Especifican la interfaz que el componente proporciona de forma independiente al lenguaje de programación o del sistema operativo donde se usará. Normalmente se inspiran en la sintaxis usada por las DLL, muy similar a la usada en las llamadas a funciones en C++.
- Invocación. Soporte para cargar y ejecutar los componentes cuando sean necesarios, enviando y recibiendo eventos, y comunicándose con otros componentes y el resto de la aplicación a través de la red.
- Introspección. Una forma de aislar el componente del mundo exterior, lo que permite centrarse solamente en lo que hace y cómo lo hace.
- Distribución. Los componentes pueden transmitirse a través de una red.

La explosión de la tecnología de componentes fue lenta, en parte debido al sentimiento de insatisfacción que la orientación por objetos dejó en sus inicios: las promesas de facilidad en la reutilización de código nunca fueron cumplidas, hasta ahora.

CREAR LA ARQUITECTURA DE UNA EMPRESA

Diseñar la arquitectura de sistemas de una gran organización es una tarea que puede resultar intimidatoria a mucha gente; son tres los requisitos para empezar: un departamento de arquitectura, la redacción de un anteproyecto de diseño de la arquitectura y un documento que recoja los valores que impulsará. Es fácil de olvidar, dado el alcance actual, que el uso de la informática en las organizaciones es un fenómeno relativamente reciente. No es de extrañar que las estructuras y procesos, para obtener un valor real de esta inversión, hayan retrasado la inversión en sí misma, y el resultado fue un gigantesco enredo.

La tónica aplicada hasta el momento por la mayoría de las empresas es buscar la robustez de los estándares, a través de compra de tecnología a un único fabricante o exigiendo el desarrollo de aplicaciones que funcionen sobre una rígida arquitectura. Lo primero es crear la dirección de arquitectura, que coordine y se asegure de que la arquitectura del sistema facilita a las aplicaciones existentes -y futuras- la consecución de los objetivos de la empresa. Las tareas del departamento de arquitectura implican: amplia visión de la actividad de la organización, estar al día de los más relevantes progresos tecnológicos y asegurarse de que los equipos que trabajan en el desarrollo de aplicaciones cumplen las directivas oportunas. El jefe del departamento de arquitectura debe jugar entre el punto de vista del empresario y el

del técnico; debe ser pragmático antes que idealista; evangelista antes que dictador; y debe, por supuesto, ser de la absoluta confianza del director de sistemas de la organización.

Para comprender donde no se debe de meter el departamento de arquitectura, volvamos al modelo de cuatro capas de la organización: los desarrolladores son sus clientes y los responsables de la infraestructura sus proveedores. Los arquitectos de sistemas que pasan demasiado tiempo eligiendo sistemas operativos o rediseñando las líneas de negocio de su organización, no invierten el tiempo necesario en su trabajo.

De cualquier modo, la línea de división entre infraestructura y arquitectura puede ser muy difícil de ver: muchos de los elementos que en un momento dado forman parte de la arquitectura, posteriormente se consideran parte de la infraestructura, como ocurre con las redes locales y ocurrirá con los sistemas de mensajería. En cualquier caso, es responsabilidad del jefe de arquitectura promover este tipo de progresos, porque como es lógico, con una débil infraestructura no es posible edificar una sólida arquitectura.

Una importante clave en el enfoque moderno de la informática es ver las aplicaciones como una colección de servicios. Estos servicios deberían ser, hasta donde sea posible, independientes unos de otros, de forma que se puedan sustituir, eliminar o añadir nuevas funcionalidades sin demasiado esfuerzo. Separar de esta forma los servicios proporciona el valor añadido de poder elegir la mejor tecnología para cada uno de ellos: los datos de los clientes en una base de datos relacional, los de los empleados en un directorio que cumpla las especificaciones LDAP y las descripciones de los productos en un servidor web.

El problema en este idílico paisaje es que la mayoría de los servicios con las que ya cuenta la organización no están correctamente empaquetados y con sus interfaces bien definidas, sino que se encuentran sepultados bajo aplicaciones existentes, en paquetes cerrados de sistemas propietarios. Por esto es necesario un anteproyecto del sistema que se desea tener, de forma que se pueda trazar un camino para llegar hasta él. Cualquier nuevo diseño debe responder a dos preguntas: de qué servicios existentes se puede hacer uso y a qué nuevo servicio se puede contribuir. Esto supone no volver a pensar en aplicaciones aisladas, para comenzar a pensar en términos de compartir servicios de aplicaciones.

El último punto es crear un documento con los valores de la arquitectura. Debe ser un documento breve, de menos de 1000 palabras, distribuido a todo el personal de sistemas de la empresa. Debe cubrir puntos como: definir cuando se deben usar dos capas y cuando tres; qué *middleware* se usará y que estándares -sistemas operativos, de mensajería, etc.- no están abiertos a debate; ser un documento deliberadamente minimalista, permitiendo libertad a la gente con creatividad para elegir cuál es la mejor solución para resolver su problema particular.

Es necesario concentrarse en tres ventajas fundamentales que este documento debe aportar: 1) elimina los debates estériles al señalar claramente qué puntos no están abiertos a debate, de forma que el personal se centre en debatir los problemas reales; 2) ayuda a identificar al personal que cumple los mejores requisitos para trabajar con la arquitectura, de forma que se puede utilizar este documento como criterio de selección de los nuevos técnicos; y 3) le proporciona a los técnicos la posibilidad de enfocar sus esfuerzos y su creatividad en campos en los que realmente serán apreciados.

A continuación se muestra un ejemplo muy simple a partir del cual se podría empezar a trabajar:

Este documento, que muestra nuestra posición frente a los más relevantes puntos de la arquitectura de sistemas de esta empresa, fue redactado por [el jefe de arquitectura] y ratificado por [el director de sistemas]. Fue revisado por última vez en [fecha] y, si ha transcurrido más de un año, probablemente tienes en tus manos una versión obsoleta.

Este documento se redactó con la idea de facilitar el trabajo a todo el personal de sistemas de [nombre de la organización] y no para censurarlo ni limitarlo. Si tienes una razón lo suficientemente fuerte como para contravenir

alguno de los puntos aquí indicados, documéntala y hazla llegar.

1. *Nunca olvides que la actividad de nuestra empresa está encaminada a [actividad de la empresa] y que en el departamento de sistemas nuestra principal prioridad es desarrollar sistemas que soporten esa línea de negocios. Dejemos a otros la innovación en la tecnología para que nosotros innovemos en cómo aplicar esa tecnología en una aplicación real.*
2. *Debemos maximizar el valor de nuestro trabajo desarrollando servicios para aplicaciones en lugar de aplicaciones aisladas, permitiéndonos reutilizar en el futuro esos servicios para crear nuevas aplicaciones rápida y fácilmente. Nuestro anteproyecto de servicio de aplicaciones define los servicios que necesitamos y si existen o no en este momento. Cuando diseñes nueva aplicaciones te pedimos que uses todos los servicios existentes que necesites y consideres a qué nuevos servicios puede contribuir tu aplicación.*
3. *En el nivel más bajo pensamos que todas las aplicaciones deben de estar ensambladas por componentes software. La tecnología que uses para ello no es tan importante, pero a menos que tengas una buena razón para no hacerlo preferimos que uses [tecnología de componentes preferida, por ejemplo ActiveX] porque [motivos para preferirla, por ejemplo, parece que dominará el mercado durante algunos años].*
4. *Deseamos que las herramientas y tecnologías basadas en Web sean estándares en nuestra empresa de aquí a dos años. Considera esto y realiza una aproximación siempre que sea posible.*
5. *Usa tres capas para todas las aplicaciones no triviales -por ejemplo, las entradas simples de datos podrían ir sobre dos capas. Esto facilita el mantenimiento y la reutilización del código y mejora el rendimiento de las aplicaciones.*
6. *Para aplicaciones basadas en objetos distribuidos, preferimos [nombre de la tecnología, por ejemplo CORBA]. Valoramos el tiempo de desarrollo por encima del purismo.*
7. *Excepto para algunos informes especiales, no necesitamos recoger datos de ningún tipo. Existe una estrategia especial de Data Warehouse que se encarga de ello.*
8. *Si puedes encontrar un paquete desarrollado que realice el 80% de la funcionalidad que buscamos úsalo, en otro caso considera un desarrollo a medida: la excesiva personalización de un paquete conlleva demasiados riesgos. Los paquetes que no son lo suficientemente abiertos para jugar su papel de colaboración en nuestro anteproyecto de servicios de aplicaciones, deberían ser rechazados.*
9. *Esperamos que la comunicación entre aplicaciones se realice mediante RPC y productos de mensajería.*
10. *Toma como inamovible la actual infraestructura. En ella incluimos Windows 7 y Office 2007. No desarrolles nada sobre entornos de 32 bits. Utiliza siempre entornos de 64 bits.*
11. *Considera la inversión en personal específico cuando investigues nuevas herramientas. Para la mayoría de las herramientas, esta inversión excede cualquier otra diferencia técnica de las mismas. Esto significa que tenemos una serie de preferencias: Oracle para bases de datos relacionales, SQL vía ODBC para el acceso a los datos, Microsoft Windows Server como servidores, C++ o Java como herramientas de desarrollo de tercera generación y .NET como herramienta de cuarta generación.*
12. *Evita reemplazar aplicaciones actuales a menos que sea totalmente Indispensable.*

LA ARQUITECTURA MODERNA

Publicar un conjunto de valores para una arquitectura requiere que estar bien informados. El punto de vista de un jefe de arquitectura debe contemplar los siguientes puntos:

- Estándares**.** El desarrollo de Internet y todas las tecnologías que van con ella está revolucionando el mundo de la informática. Algunos de sus efectos son rápidamente visibles, pero otros no lo son tanto. El jefe de arquitectura debe de estar bien informado de los estándares emergentes. En algunos casos aparecen problemas serios a la hora de tomar partido por dos estándares que, aparentemente, poseen la misma funcionalidad; por ejemplo DCOM y CORBA. La decisión debe tomarse recopilando información claramente imparcial y, si no es posible o no es lo suficientemente aclaratoria, deben probarse. Lo peor que se puede hacer es no usar ninguno de los dos.

- La capa intermedia**.** Ya se vio que uno de los grandes progresos de la arquitectura moderna es la subdivisión del software. El "pegamento" que une estas partes para formar una aplicación completa, es lo que se conoce como *middleware*. Por ejemplo, dos programas A y B corriendo separadamente cada uno en una máquina:

A llama a B con un problema; B trabaja en la resolución del mismo y cuando acaba devuelve la respuesta a A. El *middleware* es el que permite esta funcionalidad, pero existen algunos problemas derivados de esta forma de actuación ¿qué debería hacer el proceso A mientras que B trabaja en la resolución de su problema? ¿esperar? ¿Cómo puede B comunicarle a A algo a menos que éste lo requiera? ¿Cómo puede B saber dónde se encuentra A? ¿Qué hace A si B se cae? ¿Si B es reemplazado? ¿Si cientos de A's requieren una respuesta de una única instancia de B? Como vemos, los *middlewares* deben resolver complejas situaciones pero sin añadir excesiva complejidad a la arquitectura.

Existen dos tipos esenciales de *middlewares*: los basados en Remote Procedure Calls RPC y los basados en Message Oriented Middleware MOM. *Middlewares* basados en MOM son inherentemente asíncronos y lo más difícil en ellos es definir correctamente la estructura de los mensajes. Los *middlewares* basados en RPC son más sencillos de usar, puesto que la sintaxis de las llamadas es prácticamente idéntica a la de una llamada en C, pero el rendimiento de estos sistemas suele ser más pobre. Existen algunos fabricantes que distribuyen productos capaces de funcionar en uno u otro modo. La mejores herramientas de hoy están basadas en CORBA, están orientadas a mensajes y tienen como inconveniente que son más difíciles de usar por los programadores. Las herramientas de Microsoft basadas en COM+ son más limitadas pero muy sencillas de usar y son recomendables si anteponemos esta característica a la longevidad y la calidad del producto.

- Estructura cliente/servidor a dos o tres capas**.** La estructura cliente/servidor a dos capas nació el día en que alguien conecto su PC a una máquina UNIX. A los usuarios les gusta la facilidad de uso de los PC y a los administradores la seguridad que les reporta un servidor UNIX. Nadie lo llamó entonces arquitectura a dos capas porque no existía ninguna otra. La novedad de contar con una interfaz gráfica en lugar de una pantalla verde en modo texto fue bien recibida. Los desarrolladores comenzaron entonces a enriquecer sus productos con nuevas funcionalidades a sus productos, gracias a las mejores herramientas de desarrollo existentes para los PC. Los clientes "engordaron" haciéndose más pesados y lentos. La alternativa, meter parte de esta nueva funcionalidad en el backend no era demasiado atractiva, así que se buscó la solución introduciendo una tercera capa central, situada entre el cliente y el servidor, para sostener gran parte del peso de esta nueva funcionalidad.

  Pese a sus evidentes ventajas, los sistemas en tres capas tardaron en generalizarse debido a que son mucho más caros y difíciles de desarrollar. La arquitectura cliente/servidor a dos capas sigue siendo útil para la mayoría de los casos y ahora aparece la tecnología web para acercar la arquitectura a tres capas a las masas.

- La web**.** La tecnología web cambió sustancialmente la forma en que las aplicaciones se desarrollan. Pero la web tiene muchas más cosas que ofrecer que meramente la apariencia externa: Internet pronto conectará a la totalidad de computadores, lo que cambiará por completo las reglas del desarrollo de aplicaciones; las organizaciones que permanezcan aisladas no podrán beneficiarse de estas grandes ventajas: clientes -actuales y potenciales-, empleados y proveedores estarán continuamente conectados. La web popularizó también un gran conjunto de tecnologías actuales y pasadas -HTML, TCP/IP, Java, etc. Los arquitectos tenemos ahora un mayor conjunto de herramientas para jugar.

  Veamos un caso típico: una herramienta cliente podría consistir simplemente en una amistosa interfaz HTML utilizable mediante un navegador. Este cliente se conecta a través de un servidor web con una capa intermedia formada por componentes escritos en diversos lenguajes -C++, Visual Basic y Java- y empaquetados con *Actives* o *Java Beans*. Todo ello se sustenta sobre un servidor de transacciones -como Microsoft MTS o TP Tuxedo- y se conecta a una tercera capa

de aplicaciones propietarias -mediante Microsoft MQ o IBM MQSeries- y con un servidor de bases de datos vía ODBC. Todo ello permanece unido mediante un lenguaje basado en scripts como Java Scripts o Visual Basic Scripts. Arquitecturas como ésta popularizan por primera vez en la historia de la informática los beneficios de los lenguajes de componentes y la estructura a tres capas.

- Empaquetado. Utilizar tecnología de componentes empaquetados puede ahorrar años de desarrollo. Se ahorran en costos de desarrollo y mantenimiento y se logran beneficios como la robustez que proporcionan los componentes que son testeados por cientos de usuarios. Los problemas comienzan con la personalización. Habitualmente los paquetes de software no cumplen exactamente el 100% de los requisitos, y la arquitectura puede adaptarse a ellos o, lo que parece más lógico, adaptarlos a ella. Por regla general, si se encuentra un paquete que cumple el 80% de los requisitos, se debe comprar y personalizar; si los requisitos que cumple están por debajo de este porcentaje, es mucho mejor desarrollar el sistema. Las personalizaciones complejas pueden ser caras y, a menudo, inviables.

  Donde mejor se comportan los paquetes es en los procesos que son prácticamente iguales en multitud de empresas, como la facturación, o cuando lo que se quiere es personalizar el contenido y no la funcionalidad, como en sistemas de datos de gestión documental. A veces ocurre que se desea sustituir un paquete, pero parte de su funcionalidad es indispensable para otros elementos: el paquete se convierte en parte de la arquitectura. Para evitar estos indeseables efectos, se deben envolver los paquetes y realizar la integración contra este envoltorio. Esto encarece el costo de desarrollo pero, a la larga, mejora la flexibilidad de nuestro sistema.

- Envolviendo sistemas propietarios. Muchos sistemas propietarios proporcionan funcionalidades muy valiosas en determinadas misiones críticas. Son fruto de años de experiencia y su uso en una arquitectura bien diseñada puede ser muy beneficioso. La forma de conseguirlo es envolverlo con una interfaz que lo haga comportarse como uno de los paquetes existentes. A pesar de usar tecnologías obsoletas, los sistemas propietarios suelen usar dos capas bien definidas: presentación lógica y almacenamiento de datos. La mejor forma de envolverla es acceder directamente a la lógica de la aplicación. Para ello la aplicación debe de proporcionarnos un API. Si esto no es posible, la segunda opción sería acceder directamente a los datos almacenados. Esto supone un perfecto conocimiento de la estructura de almacenamiento de la aplicación y, a menudo, tener que solventar problemas de sincronización y gestión de bloqueos. La última línea de ataque sería atacar el nivel de presentación. El envoltorio se comunica con la aplicación igual que lo haría un usuario. Tecleando datos y recogiendo respuestas de la salida por defecto por medio de una interfaz de terminal. Las tres técnicas son caras y difíciles, pero seguramente usar la aplicación será mucho mejor que desecharla o desarrollar otra similar.

CONCLUSIONES

Como puede abstraerse del anterior artículo, la especialización de la labor de los Ingenieros de Sistemas es cada vez más necesaria. Las funciones que antes se consideraban simples o de rutina, hoy se convierten en complejas ingenierías de intervención, lo que exige a los profesionales de la informática la adquisición de conocimientos que la educación de un pregrado no le brinda.

De acuerdo con esta visión es necesario dividir la formación en Ingeniería de Sistemas, ciencias computacionales, Ingeniería Informática o como se llame en cada país, en dos áreas claramente establecidas por las exigencias de los actuales Sistemas de Información: Arquitectos de Sistemas e Ingenieros de Software. Los primeros se encargan de diseñar, construir y mantener la arquitectura sobre la que se soportará el *core* del negocio y el manejo de la información que circula en todos los procesos de la empresa, y los segundos se especializan en diseñar,

desarrollar y mantener los programas que transitan sobre la arquitectura y convierten los datos en información para la toma de decisiones y el soporte de la empresa.

Es el momento preciso para que se actualicen los programas de formación de ingenieros en el área de los sistemas, ya no es suficiente con tener micro-conocimientos de una cantidad de conceptos y materias a lo largo de ciclos de formación, hoy es necesaria la especialización en alguna de las mencionadas áreas, y la decisión debe tomarse con la misma prontitud y estructuración con la que los desarrollos tecnológicos se producen.

Ω